CSE-E4810 Machine Learning and Neural Networks (5 cr)

Lecture 5: Improved Learning Methods for Multilayer Perceptron Networks

Prof. Juha Karhunen

https://mycourses.aalto.fi/course/view.php?id=13086

Introduction

- Multilayer perceptron networks (MLP networks) have been extensively used in many fields due to their ability:
  1. To approximate complex nonlinear mappings directly from the input samples.
  2. To provide models for many phenomena that are difficult to handle using classical parametric techniques.
- But the learning speed of backpropagation (BP) type algorithms for MLP networks is often too slow.
- It is not surprising to see that their learning may take several hours or even days in very large problems.
- This has earlier been a major bottleneck in their application.
- Two key reasons for the slow learning of backpropagation type

algorithms are:

1. Gradient-based learning algorithms are slow, especially if stochastic gradient is used as in the basic BP algorithm.
2. All the parameters of the neural networks are tuned iteratively by using such learning algorithms.

- In Du's and Swamy's book, various tricks for speeding up backpropagation learning are discussed in Section 4.7.
- And in Section 4.8 some improved BP algorithms are discussed.
- Then in Chapter 5 other learning techniques for multilayer perceptron networks are presented.
- They use better optimization techniques than standard BP.
- Consequently, these methods usually converge clearly faster than standard BP.

- The price that must be paid for that is that their computational load can be much higher per one iteration.
- If the network is large, this may be prevent the use of these techniques.
- In our course, we consider only the Levenberg-Marquardt method discussed in Section 5.2.2 in Du's and Swamy's book.
- It is the most popular alternative to standard BP and yields often good results.
- But we discuss also a fairly new method called Extreme learning machine (ELM) which has become quite popular recently.
- ELM provides good generalization performance at extremely fast learning speed compared with standard backpropagation.
- In Du's and Swamy's book, ELM is discussed on one page only in subsection 10.6.4 without showing any formulas.

Levenberg-Marquardt algorithm

- The Levenberg-Marquardt backpropagation (LMBP) algorithm uses a simplified version of Newton's method for training MLP networks.
- Newton's method is a well-known numerical optimization technique with quadratic speed of convergence.
- It converges much faster than the steepest descent method.
- But its computational load per iteration is clearly higher.
- In the following, we only summarize Newton's optimization algorithm.
- It is discussed in the beginning of Section 5.2 in Du's and Swamy's book.
- And its further development, Gauss-Newton method is presented after this in subsection 5.2.1.

Summary of Newton's optimization algorithm

- Consider the problem of finding a vector $\mathbf{w} = [w_1, w_2, \ldots, w_N]^T$ that minimizes a given scalar valued energy function $E(\mathbf{w})$.
- In Newton's method, the components of $\mathbf{w}$ are first initialized to some random values.
- Let $\mathbf{w}(k)$ denote the estimate of the optimal vector at iteration k.
- The next estimate at iteration $k+1$ is computed from the updating formula

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{H}_k^{-1}\mathbf{g}_k \tag{1}$$

- There $\mathbf{g}_k$ is the gradient vector of the energy function $E(\mathbf{w})$ at iteration k:

$$\mathbf{g} = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial E(\mathbf{w})}{\partial w_N}\right]^T \tag{2}$$

- And $\mathbf{H}_k^{-1}$ is the inverse of the Hessian matrix $\mathbf{H}$ at iteration k:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E(\mathbf{w})}{\partial w_1^2} & \frac{\partial^2 E(\mathbf{w})}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_1 \partial w_N} \\ \frac{\partial^2 E(\mathbf{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 E(\mathbf{w})}{\partial w_2^2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_2 \partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 E(\mathbf{w})}{\partial w_N \partial w_1} & \frac{\partial^2 E(\mathbf{w})}{\partial w_N \partial w_2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_N^2} \end{bmatrix} \quad (3)$$

- Note that both the gradient vector (2) and the Hessian matrix (3) must be re-evaluated at each iteration (1) for the k:th estimate $\mathbf{w}(k)$.

Derivation of the Levenberg-Marquardt algorithm

- A problem with Newton's algorithm is the computational cost of calculating the inverse of the Hessian matrix.
- This limits its practical use for MLP neural networks.

- The LMBP has approximately the same convergence speed but it is significantly less complex.
- To derive this algorithm, the problem of training MLP network is first formulated as a nonlinear optimization problem.
- Consider again the MLP network in Figure 1.
- Its training can be viewed as finding optimal network weights that minimize the error between the desired and actual outputs of the network for all the Q training vector pairs.
- The corresponding energy function is

$$E(\mathbf{w}) = \frac{1}{2}\sum_{q=1}^{Q} \parallel \mathbf{d}_q - \mathbf{x}_{out,q}^{(3)} \parallel^2 = \frac{1}{2}\sum_{q=1}^{Q}\sum_{h=1}^{n_3}(d_{qh} - x_{out,q}^{(3)})^2 \tag{4}$$

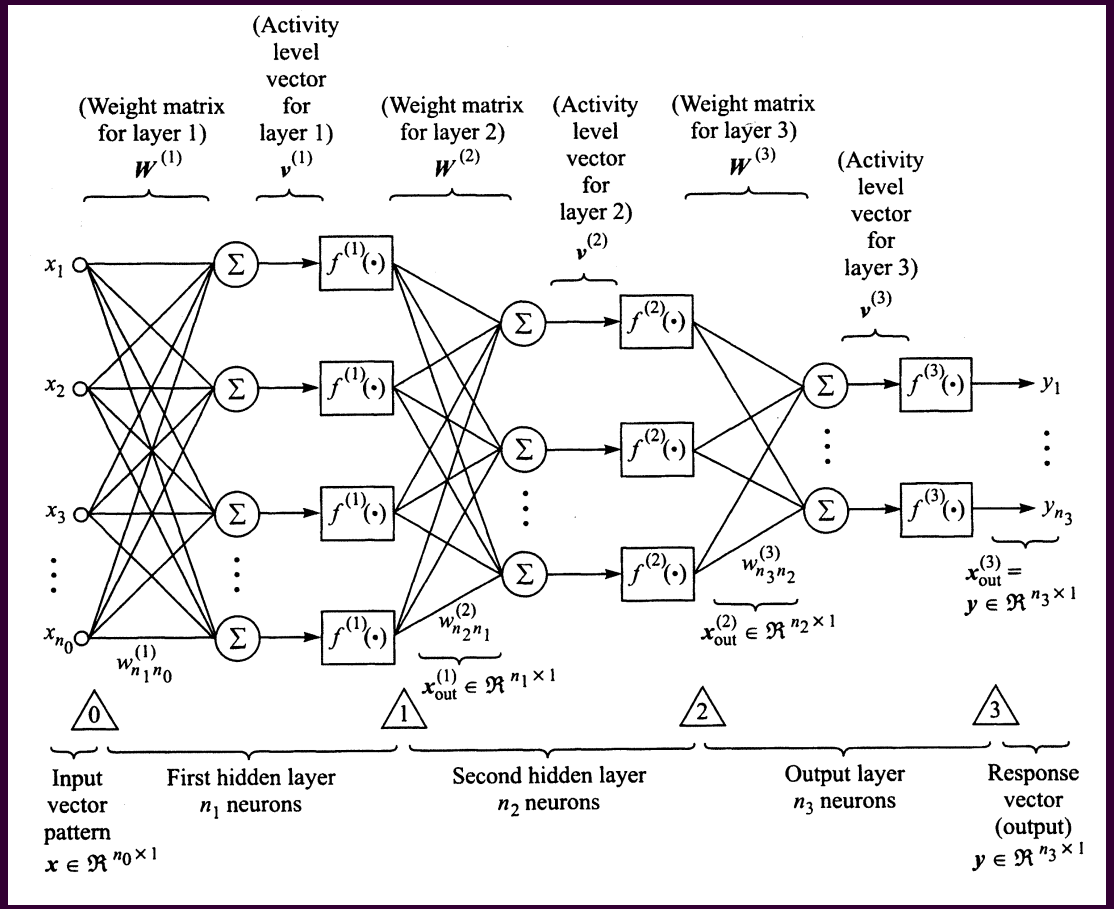

Figure 1: Notation and schematic diagram of a MLP network.

- By defining $P = n_3 Q$, (4) can be rewritten as

$$E(\mathbf{w}) = \frac{1}{2}\sum_{p=1}^{P}(d_p - x_{out,p}^{(3)})^2 = \frac{1}{2}\sum_{p=1}^{P} e_p^2 \tag{5}$$

  where

$$e_p = d_p - x_{out,p}^{(3)} \tag{6}$$

- The gradient (2) of the energy function (5) becomes

$$\mathbf{g} = \mathbf{J}^T\mathbf{e} \tag{7}$$

- There the vector

$$\mathbf{e} = [e_1, e_2, \ldots, e_P]^T \tag{8}$$

and $\mathbf{J}$ is $P \times N$ Jacobian matrix defined by

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_N} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \cdots & \frac{\partial e_2}{\partial w_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_P}{\partial w_1} & \frac{\partial e_P}{\partial w_2} & \cdots & \frac{\partial e_P}{\partial w_N} \end{bmatrix} \tag{9}$$

- Thus the element (i, j) of the Jacobian is the derivative of the i:th component of the vector $\mathbf{e}$ with respect to the j:th component of the vector $\mathbf{w}$.
- Note that the Jacobian $\mathbf{J}$ is is in general a non-square and nonsymmetric matrix.
- Note also that the i:th row of the Jacobian matrix is the gradient vector $\partial e_i / \partial \mathbf{w}$ transposed.

- The Hessian (3) of the energy function (5) becomes

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mathbf{S} \tag{10}$$

- There the $N \times N$ matrix $\mathbf{S}$ consists of second-order derivatives, see Eq. (5.12) in Du's and Swamy's book.
- When approaching the minimum of the energy function $E(\mathbf{w})$, the elements of the matrix $\mathbf{S}$ become small.
- The Hessian can then be closely approximated by

$$\mathbf{H} \approx \mathbf{J}^T\mathbf{J} \tag{11}$$

- Substituting (7) and (11) into the Newton iteration (1) yields

$$\mathbf{w}(k+1) = \mathbf{w}(k) - [\mathbf{J}_k^T\mathbf{J}_k]^{-1}\mathbf{J}_k^T\mathbf{e}_k \tag{12}$$

- Here the subscript k indicates that the Jacobian matrix $\mathbf{J}_k$ and error vector $\mathbf{e}_k$ must be evaluated for the current weight vector $\mathbf{w}(k)$.

- A problem with the iteration (12) is that it requires the inversion of the matrix $\mathbf{J}^T\mathbf{J}$.
- This can be ill-conditioned or even singular with no inverse in the worst case.
- The problem can be easily resolved by adding to $\mathbf{J}^T\mathbf{J}$ a matrix $\mu\mathbf{I}$.
- There μ is a small positive number and $\mathbf{I}$ is $N \times N$ identity matrix.
- This modification yiels the final Levenberg-Marquardt algorithm or method for updating the network weights:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - [\mathbf{J}_k^T\mathbf{J}_k + \mu_k\mathbf{I}]^{-1}\mathbf{J}_k^T\mathbf{e}_k \tag{13}$$

- This algorithm is also called the Gauss-Newton method, sometimes without the regularizing term $\mu_k\mathbf{I}$.
- It represents a transition between the steepest descent method and Newton's method.

- When μ_k is small, it approximates Newton's method.
- When μ_k is large, the second term inside the square brackets in (13) becomes dominant.
- The update equation (13) can then be approximated as

$$\mathbf{w}(k+1) \approx \mathbf{w}(k) - \frac{1}{\mu_k}\mathbf{J}_k^T\mathbf{e}_k = \mathbf{w}(k) - \alpha_k\mathbf{g}_k \tag{14}$$

  where $\mathbf{g}_k$ is the gradient (7) at iteration k, and $\alpha_k = 1/\mu_k$.
- In this case, the Levenberg-Marquardt method reduces to the steepest descent gradient method.
- The biggest problem in implementing the derived LMBP algorithm is the calculation of the Jacobian matrix $\mathbf{J}(\mathbf{w})$ defined in (9).
- Its each element has the form $J_{i,j} = \partial e_i/\partial w_j$.
- The simplest approach to estimate these partial derivatives is to use

the approximation

$$J_{i,j} \approx \frac{\Delta e_i}{\Delta w_j} \tag{15}$$

- There Δe_i is the change in the output error due to the small perturbation Δw_j of the weight w_j.
- This method is straightforward and simple to implement.
- The perturbations Δw_j of the weights are kept small, at least an order of magnitude smaller than current learning rate parameter μ_k.
- Remarks on the properties of the Levenberg-Marquardt method and some modified version of it are discussed in section 5.2.2 in Du's and Swamy's book.

Extreme learning machine

Background and theory

- Extreme learning machine (ELM) was introduced fairly recently.
- In the paper G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications", *Neurocomputing*, Vol. 70, 2006, pp. 489–501.
- In this course, it is discussed on the following slides.
- You can find a more complete theoretical and experimental discussion of ELM in the above paper if necessary.
- Extreme learning machine uses a single hidden layer feedforward neural network (SLFN).
- The weights between the input layer and the neurons of the hidden layer are chosen randomly beforehand.

- Similarly, the bias terms of the hidden layer neurons are chosen randomly beforehand.
- Such hidden neurons or nodes can be called random hidden neurons or nodes.
- The output layer is taken linear for simplicity.
- With such choices, SLFNs can be simply considered as a linear system.
- The output weights linking the hidden layer to the output layer can then be determined analytically.
- From the least-squares (pseudoinverse) solution of the hidden layer output matrices.
- This learning method is called by its authors extreme learning machine because it has the following properties:

  - Its implementation is easy.
  - It tends to reach the smallest training error.
  - It has good generalization performance.
  - It runs extremely fast compared with standard backpropagation.
- The good generalization performance of ELM follows from the fact that its solution has the smallest norm of weights.
- This in turn is the general property of the standard least-squares method.
- It has been shown theoretically that the smaller the norm of weights in a feedforward network is, the better generalization capability the network tends to have.
- Provided that it has a relatively small training error.

- The extreme learning machine is based on the following theoretical result proved in the paper mentioned above:
  - A single-layer feedforward network with N randomly chosen hidden neurons can exactly learn N distinct observations (input-output pairs).
  - Provided that the activation functions in the hidden layer are infinitely differentiable.
- The infinitely differentiable activation functions include the sigmoidal functions, sine, cosine, and exponential functions.
- As well as the radial basis functions discussed in Lecture 8.
- Later on, a stronger result was proved stating that extreme learning machines have the universal approximation property.

Mathematical formulation of ELM

- Assume that we have N training pairs $\{\mathbf{x}_i, \mathbf{t}_i\}$, $i = 1, 2, \ldots, N$, of data to be modeled at disposal.
- There $\mathbf{x}_i$ is the i:th input column vector of dimension n.
- And $\mathbf{t}_i$ is the corresponding i:th target column vector of dimension m.
- That is, $\mathbf{t}_i$ is the desired response (output) of the SLFN for $\mathbf{x}_i$.
- Thus in the SLFN considered there are n neurons in the input layer and m neurons in the linear output layer.
- In the single hidden layer, there are M neurons.
- Usually the number M of neurons in the hidden layer is smaller than the number N of training data pairs.
- The weight vector of j:th hidden layer neuron is denoted by $\mathbf{w}_j$.

- It is an n-dimensional column vector containing the scalar weights between the neurons (elements of a data vector) in the input layer and neuron j in the hidden layer.
- In ELM, these weight vectors $\mathbf{w}_j$, $j = 1, 2, \ldots, M$ are chosen randomly before learning.
- Also the scalar bias terms β_j of the hidden layer neurons are chosen randomly before learning.
- Thus the output of the j:th neuron in the hidden layer corresponding to the input vector $\mathbf{x}_i$ is

$$g_j(\mathbf{x}_i) = g(\mathbf{w}_j^T \mathbf{x}_i + \beta_j) \tag{16}$$

- There $g(t)$ is the chosen nonlinear activation function, for example the sigmoidal $\tanh(t)$ function suitably scaled.
- The total output vector $\mathbf{o}_i$ of the extreme learning machine

corresponding to the input vector $\mathbf{x}_i$ is

$$\sum_{j=1}^{M} \mathbf{b}_j g_j(\mathbf{x}_i) = \sum_{j=1}^{M} \mathbf{b}_j g(\mathbf{w}_j^T \mathbf{x}_i + \beta_j) = \mathbf{o}_i \tag{17}$$

- There $\mathbf{b}_j$ is the m-dimensional column vector containing the m weights from the j:th neuron in the hidden layer to the m neurons in the output layer.
- The output vectors $\mathbf{o}_i$ are equated to the target vectors $\mathbf{t}_i$ corresponding to the input vector $\mathbf{x}_i$.
- Thus we get the vector equation

$$\sum_{j=1}^{M} \mathbf{b}_j g(\mathbf{w}_j^T \mathbf{x}_i + \beta_j) = \mathbf{t}_i \tag{18}$$

for each training pair $\{\mathbf{x}_i, \mathbf{t}_i\}$, $i = 1, 2, \ldots, N$.

- These N vector equations can be expressed compactly as the matrix equation

$$\mathbf{HB} = \mathbf{T} \tag{19}$$

- There $\mathbf{T}$ is the $N \times m$ target matrix whose i:th row vector is $\mathbf{t}_i^T$.
- And the matrix $\mathbf{B}$ is $M \times m$ weight matrix containing the weights between the hidden layer and output layer.
- The j:th row vector of the matrix $\mathbf{B}$ is $\mathbf{b}_j^T$.
- It contains the weights from j:th hidden neuron to the output layer.
- Finally, the $N \times M$ matrix $\mathbf{H}$ is called the hidden layer output matrix of the neural network.
- The element h_{ij} on the i:th row and j:th column of $\mathbf{H}$ is $g(\mathbf{w}_j^T \mathbf{x}_i + \beta_j)$.
- The j:th column vector of the matrix $\mathbf{H}$ contains the outputs of the

j:th hidden neuron with respect to the input vectors $\mathbf{x}_i$, $i = 1, 2, \ldots, N$.

- And the i:th row vector of the matrix $\mathbf{H}$ contains the outputs of all the M neurons in the hidden layer with respect to the input vector $\mathbf{x}_i$.
- Note that the matrix $\mathbf{H}$ depends on all the M weight vectors $\mathbf{w}_j$ and bias terms β_j, $j = 1, 2, \ldots, M$, of the M hidden layer neurons.
- And on all the N input vectors $\mathbf{x}_i$, $i = 1, 2, \ldots, N$.
- But since we know all these quantities, we can insert them and compute the constant value of the matrix $\mathbf{H}$.
- Similarly, the matrix $\mathbf{T}$ consisting of target vectors is known constant matrix.
- Thus the only unknown in the matrix equation $\mathbf{HB} = \mathbf{T}$ is the $M \times m$ weight matrix $\mathbf{B}$.

- The formal solution of this matrix equation is

$$\mathbf{B} = \mathbf{H}^{+}\mathbf{T} \tag{20}$$

  where $\mathbf{H}^{+}$ is the pseudoinverse of the matrix $\mathbf{H}$.
- Pseudoinverse is a generalization of standard matrix inverse for rectangular matrices.
- The pseudoinverse $\mathbf{H}^{+}$ provides the minimum norm solution (20) of the matrix equation $\mathbf{HB} = \mathbf{T}$ in different cases.
- Note that in this matrix equation there are $M \times m$ unknown values, namely elements of the $M \times m$ output weight matrix $\mathbf{B}$.
- And $N \times m$ equations.
- Consider first the case $M = N$, in which there are as many neurons in the hidden layer as training data pairs.
- The $N \times M$ matrix $\mathbf{H}$ becomes then a square matrix, and the

optimal weight matrix is simply

$$\mathbf{B} = \mathbf{H}^{-1}\mathbf{T} \tag{21}$$

provided that the matrix $\mathbf{H}$ is nonsingular.

- Consider then the realistic case $M < N$, where there are less neurons in the hidden layer than training data pairs.
- In the matrix equation $\mathbf{HB} = \mathbf{T}$ there are then more equations than unknowns.
- The equations do not have then usually any solution, but the pseudoinverse solution is then defined by

$$\mathbf{B} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T} \tag{22}$$

- This is the familiar least-squares solution which minimizes the squared error in solving the matrix equation.

- If the matrix $\mathbf{H}^T\mathbf{H}$ is almost singular, you can regularize the least-squares solution (22) by replacing it with

$$\mathbf{B} = (\mathbf{H}^T\mathbf{H} + \mu\mathbf{I})^{-1}\mathbf{H}^T\mathbf{T} \tag{23}$$

  where μ is a small constant and $\mathbf{I}$ unit matrix.

- We shall discuss regularization more in the next lecture.
- In general, the regularized solution can be derived by minimizing with respect to the matrix $\mathbf{B}$ the criterion

$$C = \frac{1}{2} \parallel \mathbf{B} \parallel^2 + \frac{1}{2\mu} \parallel \mathbf{T} - \mathbf{HB} \parallel^2 \tag{24}$$

- If the regularizing parameter μ is large, we weight relatively less the data expressed in the equation $\mathbf{T} = \mathbf{HB}$.
- And more the minimization of the norm $\parallel \mathbf{B} \parallel^2$ of the weight matrix $\mathbf{B}$, and vice versa.

- In the case $M > N$ there are more neurons M in the hidden layer than training pairs N.
- Then the training error can be made zero, but somewhat surprisingly ELM can perform well also in this case.
- This can be achieved by considering the regularized pseudoinverse solution for the criterion (24), which is for the case $M > N$

$$\mathbf{B} = \mathbf{H}^T(\mu\mathbf{I} + \mathbf{H}\mathbf{H}^T)^{-1}\mathbf{T} \tag{25}$$

- One should compute the solution (25) for a large range of regularizing parameters $\mu = 10^{-8}, 10^{-7}, \ldots, 10^{7}, 10^{8}$.
- And the select the solution which minimizes the validation error for data that has not been used in the training phase.
- The same procedure can be applied when $M < N$ in the equation (23).

A summary of the ELM learning algorithm

- We now summarize the extreme learning machine method for learning a single hidden layer feedforward network.
- Assume that we have a training set $\{\mathbf{x}_i, \mathbf{t}_i\}$, $i = 1, 2, \ldots, N$.
- There $\mathbf{x}_i$ the i:th input vector of dimension n, and $\mathbf{t}_i$ is the corresponding target vector of dimension m.
- Choose the activation function $g(t)$ and number M of neurons in the hidden layer. Then
  1. Assign randomly the hidden layer weight vectors $\mathbf{w}_j$ and biases β_j, $j = 1, 2, \ldots, M$.
  2. Calculate the $N \times M$ hidden layer output matrix $\mathbf{H}$. Its elements are $h_{ij} = g(\mathbf{w}_j^T \mathbf{x}_i + \beta_j)$.

3. Calculate the $M \times m$ weight matrix $\mathbf{B}$ of the output layer from

$$\mathbf{B} = \mathbf{H}^{+}\mathbf{T} \tag{26}$$

where the $m \times N$ target matrix $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_N]^T$.

- ELM has several advantages compared with standard backpropagation or its improvements as mentioned before.
- The main drawbacks of the ELM are:
  - If trained properly, MLP networks with more than one hidden layer used in ELM can possibly achieve somewhat better results.
  - For achieving comparable results, the number of neurons in ELM's hidden layer must often be much larger than in standard backpropagation and its improvements.
  - Like other NN methods ELM may suffer from overfitting if the number of neurons in the hidden layer is chosen too large.

- The name extreme learning machine promises too much.
- Similar ideas on selecting randomly weights have been introduced earlier by other authors but then largely forgotten.

Some experimental results

- In the paper where ELM is introduced lots of experimental results are presented, confirming its good performance.
- For real-world data sets whose complexity varies greatly, collected from different fields.
- Both regression and classification problems are considered.
- Regression means simply modeling an n-dimensional vector variable $\mathbf{y}$ in terms of another m-dimensional vector variable $\mathbf{x}$ as

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \tag{27}$$

- In neural networks, regression is usually equivalent to modeling the unknown input-output mapping $\mathbf{f}(\cdot)$ by a neural network.
- That is, function approximation by the chosen neural network.
- In the experiments, the input vectors $\mathbf{x}_i$ are preprocesssed by normalizing the values of their components into the range $[0, 1]$.
- And the components of the target vectors $\mathbf{t}_i$ are normalized into the range $[-1, 1]$.
- The activation function is the simple sigmoidal function

$$g(t) = \frac{1}{1 + e^{-t}} \tag{28}$$

- Extreme learning machine is compared with the Levenberg-Marquardt algorithm and support vector machines.
- Levenberg-Marquardt algorithm is chosen to represent

backpropagation type algorithms.

- Because it appears to be the fastest method for training moderate sized MLP networks (up to several hundred weights).
- Support vector machines will be discussed in Lecture 9.
- They have also one hidden layer.
- In these experiments, the nonlinearities used in support vector machines are radial-basis functions.
- A typical example of radial-basis functions (see Lecture 8) is Gaussian function.
- Consider first an artificially generated data set.
- Such data sets are useful in testing the performance of various methods because the correct results are known.

- The task is to approximate the sinc function

$$y(x) = \begin{cases} \sin(x)/x, & x \neq 0, \\ 1, & x = 0 \end{cases} \tag{29}$$

- Both the training set and test set contained 5000 sample pairs (x_i, y_i).
- There the scalar inputs x_i are uniformly distributed random numbers in the interval $(-10, 10)$.
- To make the regression problem more realistic, large uniformly distributed noise in the interval $[-0.2, 0.2]$ was added to all the training samples y_i.
- But the test data set remained noise free.
- The results are shown in Figures 2-4.

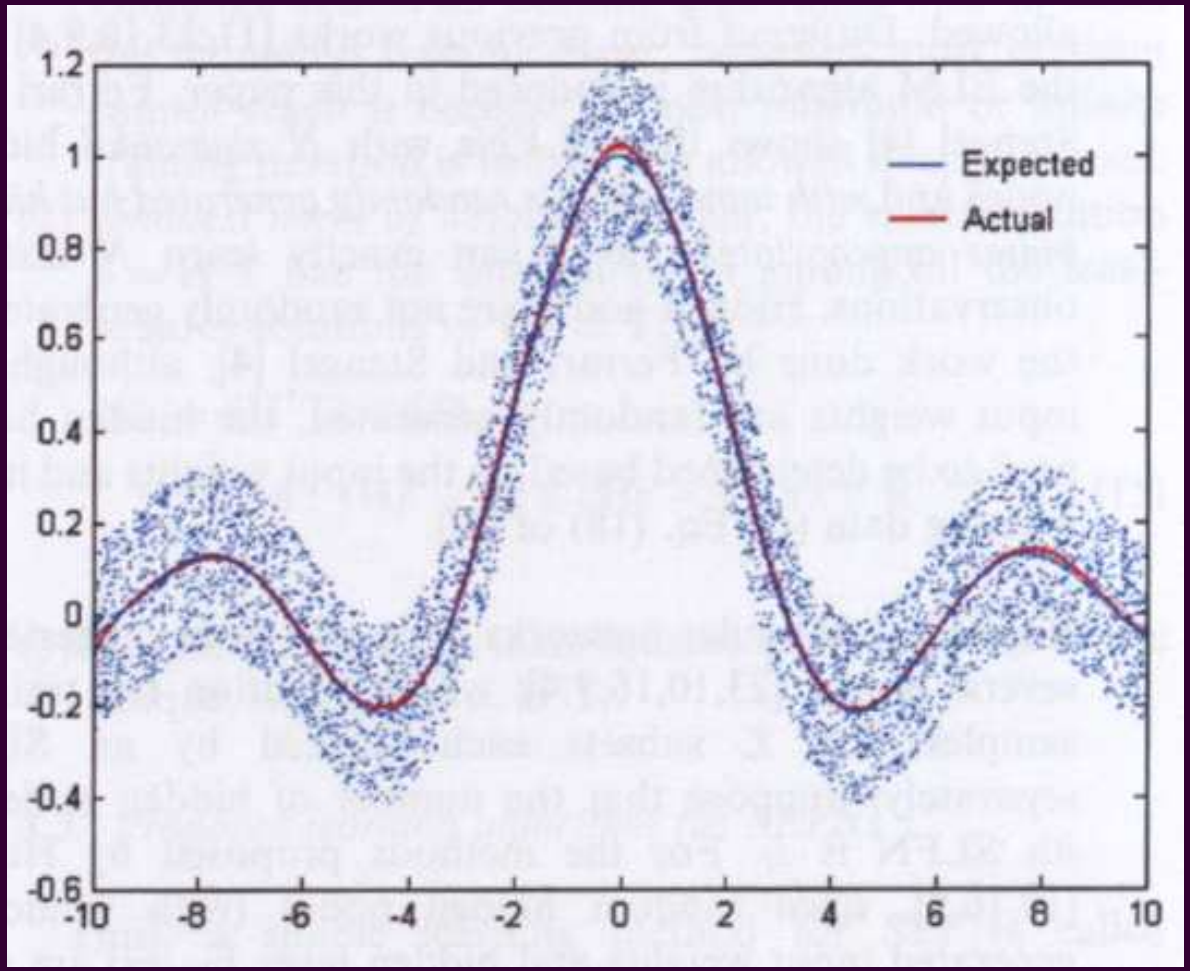

Figure 2: Outputs of the extreme learning machine method.

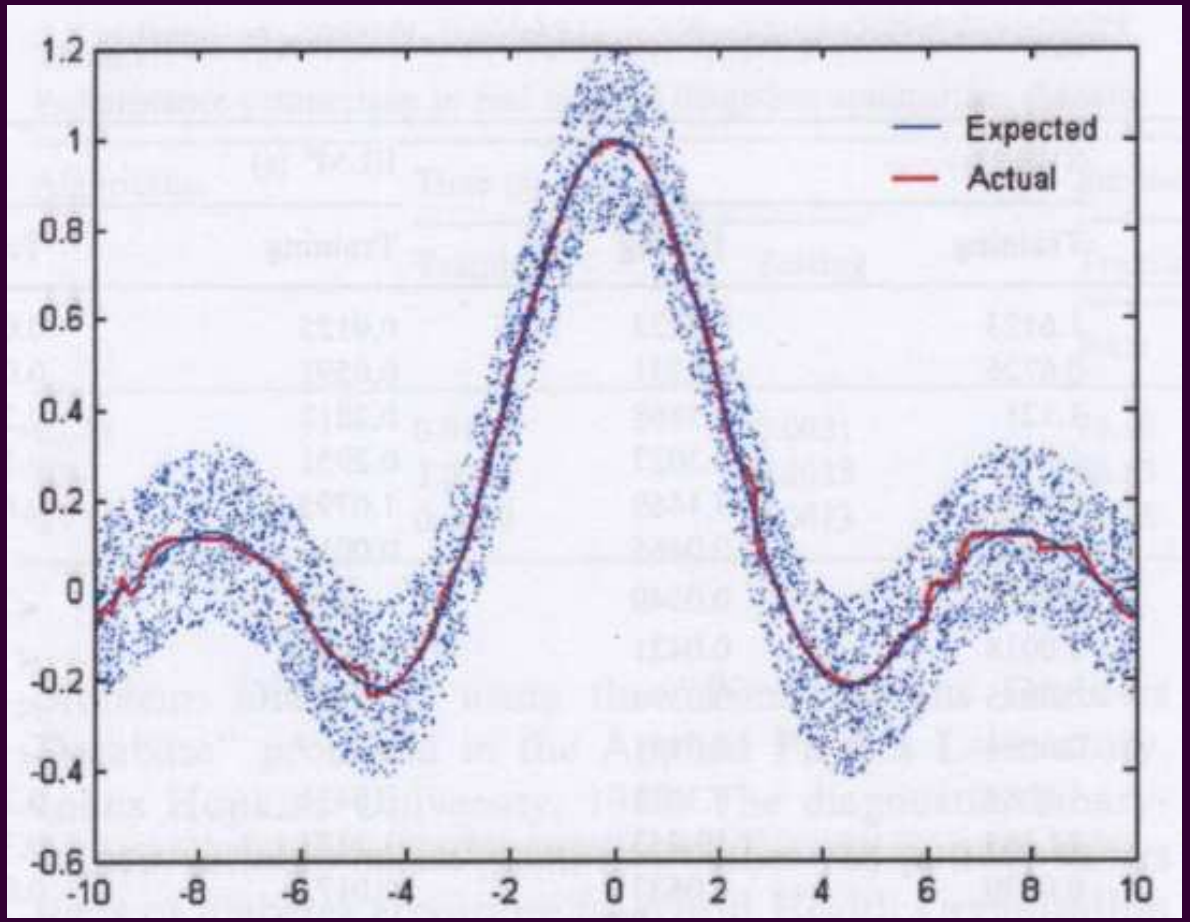

Figure 3: Outputs of the Levenberg-Marquardt backprop algorithm.

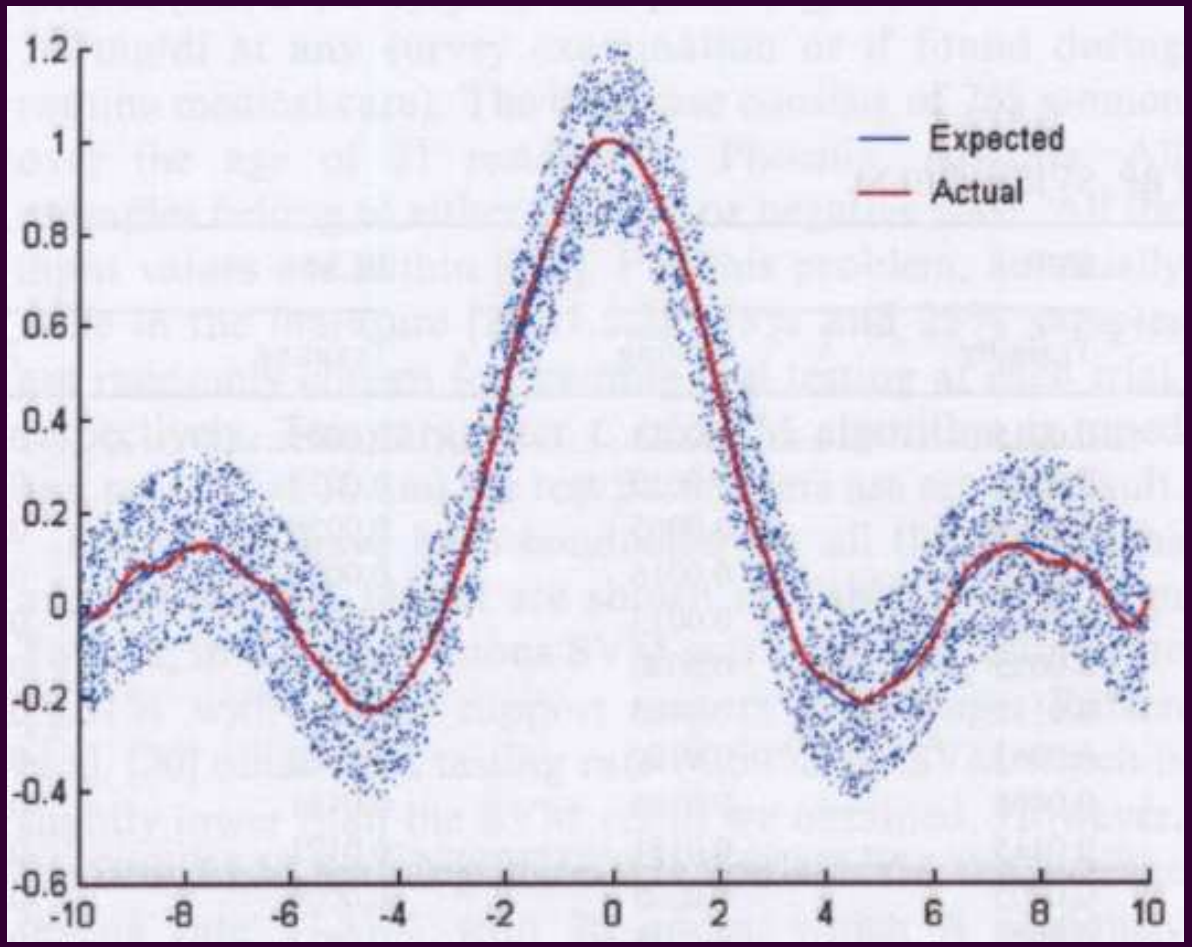

Figure 4: Outputs of the support vector regression learning algorithm.

- Both the ELM and Levenberg-Marquardt algorithm used 20 neurons in the hidden layer.
- While the number of support vector (nodes in the hidden layer) was 2500.
- The average results for 50 trials (realizations) of the sinc approximation problem are as follows.
- Learning times in seconds (CPU time):
  – 0.125 for extreme learning machine;
  – 21.26 for Levenberg-Marquardt algorithm;
  – 1273 for the support vector machine.
- The time elapsed to testing (computing the results for the test set) was:
  – About 0.03 seconds CPU time for ELM and Levenberg-Marquardt algorithm;

– But about 5.9 seconds for the support vector machine.

- The root mean-square (RMS) training error for all these three methods is about the same, 0.115 – 0.12.
- But the corresponding RMS test error is smaller for ELM than for the two other methods in this example, namely:
  – 0.0097 for the extreme learning machine;
  – 0.0159 for the Levenberg-Marquardt algorithm;
  – 0.0130 for the support vector machine.

- Recall the original paper G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications", *Neurocomputing*, Vol. 70, 2006, pp. 489–501.
- There ELM is compared with the Levenberg-Marquardt algorithm and support vector machines in several real-world problems.

- Including regression and classification problems of different size and complexity, and a very large complex classification problem.
- The performances of the three methods for the test data are roughly the same.
- However, ELM generalizes slightly or somewhat better than a MLP network trained with the Levenberg-Marquardt algorithm.
- This is obviously due to the local minimum problem in the latter method.
- But the learning time required for ELM is a small fraction only.
- Extreme learning machine requires typically 3–10 times more neurons in the hidden layer than the Levenberg-Marquardt algorithm for achieving the same performance.
- But sometimes even more, and sometimes less neurons.

Extensions and modifications of ELM

- ELM has been extended and modified in many ways.
- See http://www.ntu.edu.sg/home/egbhuang/ for references and conferences on extensions of ELM.
- Selected best papers of the ELM conferences have been published in special issues of the journal Neurocomputing.
- In Aalto University, the EIML research group has applied and improved ELM in many papers.
- See the publication page of the EIML group: http://research.ics.aalto.fi/eiml/publications.shtml
- These contributions include optimal pruning, regularization, and combining several ELMs.